Teil 2: Einführung in das Relationale Modell und SQL

Literatur:

- Elmasri/Navathe: Fundamentals of Database Systems, 3rd Edition, 1999.
  Section 7.1, “Relational Model Concepts”
  Section 8.2, “Basic Queries in SQL”
- Kemper/Eickler: Datenbanksysteme (in German), 3rd Edition, 1999.
  Section 3.1, “Definition des relationalen Modells” (“Definition of the Relational Model”)
  Section 4.6, “Einfache SQL-Anfragen” (“Simple SQL Queries”)
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Sunderraman: Oracle Programming, A Primer. Addison-Wesley, 1999.
- Oracle8i SQL Reference, Rel. 2 (8.1.6), Oracle Corp., Dec. 1999, Part No. A76989-01.
- SQL*Plus: Quick Reference, Rel. 8.1.6, Oracle Corp., Oct. 1999, Part No. 75665-01.
- SQL*Plus: User’s Guide and Reference, Rel. 8.1.6, Oct. 1999, Part No. A75664-01.
- Codd: A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387, 1970.
- Boyce/Chamberlin: SEQUEL: A structured English query language. In ACM SIGMOD Conf. on the Management of Data, 1974.
- Astrahan et al: System R: A relational approach to database management. ACM Transactions on Database Systems 1(2), 97–137, 1976.

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- einige Grundbegriffe des relationalen Modells erklären.
- einfache SQL-Anfragen schreiben (die sich nur auf eine Tabelle beziehen).
- Oracle SQL*Plus und eine Web-Schnittstelle zur Auswertung von SQL-Anfragen benutzen.

  Der Zweck dieses Kapitels ist, möglichst schnell mit den praktischen Übungen beginnen zu können. Einige Inhalte werden später noch einmal ausführlicher behandelt und vertieft.

Inhalt

1. Relationales Modell, Beispiel-Datenbank
2. Benutzung von SQL*Plus
3. Einfache SQL-Anfragen
4. Historische Bemerkungen

Das relationale Modell (1)

- Das relationale Modell strukturiert die Daten in Tabellenform. Ein relationales DB-Schema definiert:
  - ◇ Namen der Tabellen in der Datenbank.

    Ein relationales DB-Schema besteht im allgemeinen aus mehreren Tabellen.
  - ◇ Die Spalten jeder Tabelle (jeweils Spalten-Name und Datentyp).

    Jede Spalte kann nur Daten eine bestimmten Typs speichern, z.B. Zeichenketten (Strings) oder Zahlen einer bestimmten Länge und Genauigkeit, oder auch Datumswerte, etc.
  - ◇ Integritätsbedingungen.

    Das sind Bedingungen, die die Daten erfüllen müssen (s.u.).

Das relationale Modell (2)

- Z.B. wird Oracle mit einer Beispiel-Datenbank aus drei Tabellen ausgeliefert:
  - ◇ EMP: Information über Angestellte einer Firma.

    Engl. "employees".
  - ◇ DEPT: Information über Abteilungen.

    Engl. "departments".
  - ◇ SALGRADE: Information über Gehaltsstufen

    Diese Tabelle wird hier nicht verwendet. Je nach Version der Beispiel-Datenbank gibt es eventuell noch weitere Tabellen. Ggf. muß man das Kommando "demobld" eingeben, um die Beispiel-Datenbank anzulegen.

Das relationale Modell (3)

- Die Tabelle DEPT hat folgende Spalten:
  - ◇ DEPTNO: Abteilungsnummer (department number)
  - ◇ DNAME: Abteilungsname (department name)
  - ◇ LOC: Ort (location)

DEPT		
DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Das relationale Modell (4)

Die Spalten haben folgende Datentypen:

- DEPTNO hat den Typ NUMERIC(2), d.h. es ist eine zweistellige Zahl von -99 bis +99.

  Man will natürlich keine negativen Abteilungsnummern. Sie können mit einer Integritätsbedingung ausgeschlossen werden (siehe Kap. 8).
- DNAME hat den Typ VARCHAR(14), d.h. es ist eine Zeichenkette variabler Länge bis maximal 14 Zeichen.
- LOC hat den Typ VARCHAR(13).

Der System-Katalog (Data Dictionary) gibt andere Typ-Namen an: NUMBER(2), VARCHAR2(14), and VARCHAR2(13). Dies sind Oracle-spezifische Namen für die oben genannten Typen aus Standard-SQL (die Oracle auch versteht).

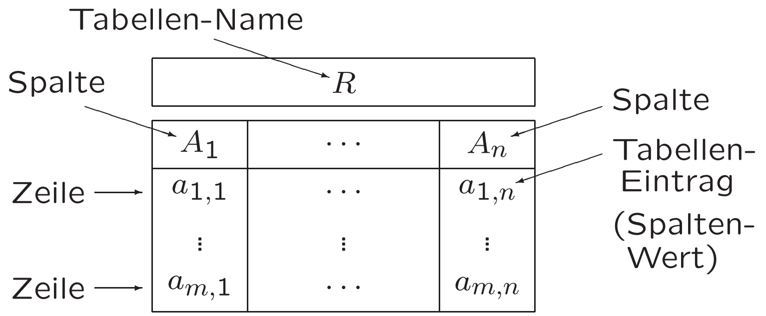
Das relationale Modell (5)

- Ein relationaler DB-Zustand (Instanz des Schemas) definiert für jede Tabelle eine Menge von Zeilen.
- Die Beispiel-Tabelle "DEPT" hat im Moment (d.h. im aktuellen Zustand) vier Zeilen.

  Das relationale Modell legt keine bestimmte Ordnung für die Zeilen fest (erste, zweite, u.s.w. Zeile). Die Speicher-Reihenfolge bestimmt das System. Zeilen können aber für die Ausgabe sortiert werden.
- Jede Zeile legt Werte für alle Spalten der Tabelle fest (kompatibel mit den deklarierten Datentypen).

  Z.B. gibt es eine Zeile mit dem Wert 10 für die Spalte DEPTNO, dem Wert 'ACCOUNTING' für DNAME, und dem Wert 'NEW YORK' für LOC.

Zusammenfassung



Statt Tabelle kann man auch Relation sagen, statt Zeile auch Tupel, und statt Spalte auch Attribut. Die Begriffe "Relation" und "Tupel" werden auf den folgenden Folien erläutert, "Attribut" in Kapitel 6.

Etwas Mathematik (1)

- Mathematisch gesehen sind die "Tabellen" des relationalen Modells Relationen. (daher der Name).
- Eine Relation ist eine Teilmenge eines kartesischen Produktes (auch Kreuzprodukt genannt: $\times$).
- Das kartesische Produkt wird z.B. auch bei (X, Y)-Koordinaten verwendet (Paare von reellen Zahlen):

$$\mathbb{R} \times \mathbb{R} = \{(X, Y) \mid X \in \mathbb{R},\ Y \in \mathbb{R}\}.$$

- Entsprechend kann man auch Tripel konstruieren:

$$\mathbb{R} \times \mathbb{R} \times \mathbb{R} = \{(X, Y, Z) \mid X \in \mathbb{R},\ Y \in \mathbb{R},\ Z \in \mathbb{R}\}.$$

Etwas Mathematik (2)

- Allgemein spricht man von n-Tupeln oder kurz Tupeln (z.B. wären Paare 2-Tupel).
- Es ist auch möglich, daß die Komponenten eines Tupels aus verschiedenen Mengen stammen. Sei
  - ◇ N_s die Menge der s-stelligen ganzen Zahlen, z.B. $N_2 := \{-99, -98, \ldots, -1, 0, 1, \ldots, 98, 99\}$.
  - ◇ V_m die Menge der Zeichenketten (Strings) bis zur maximalen Länge m,
- Dann entspricht eine Zeile aus `DEPT` einem Tripel

$$(D, N, L) \in N_2 \times V_{14} \times V_{13}.$$

Etwas Mathematik (3)

- Eine typische Relation aus der Mathematik ist $<$ auf den reellen Zahlen. Sie wird formalisiert als die Menge der $(X, Y) \in \mathbb{R} \times \mathbb{R}$ mit "X ist kleiner als Y".
- Es sind also nur bestimmte Paare in der Relation enthalten, nicht alle Paare. Daher ist $< \subseteq \mathbb{R} \times \mathbb{R}$.
- Während $<$ eine unendliche Menge von Paaren ist, sind die Relationen in relationalen Datenbanken immer endliche Mengen (von Tupeln).
- In beiden Fällen sind es aber Teilmengen eines kartesischen Produktes: `DEPT` $\subseteq N_2 \times V_{14} \times V_{13}$.

Etwas Mathematik (4)

- Im aktuellen Zustand gilt also:

```
DEPT = { (10, 'ACCOUNTING', 'NEW YORK'),
         (20, 'RESEARCH',   'DALLAS'),
         (30, 'SALES',      'CHICAGO'),
         (40, 'OPERATIONS', 'BOSTON')    }
```

- In dieser mathematischen Formalisierung werden die Spalten über ihre Position indentifiziert.

  Man kann sich zusätzlich merken, daß DEPTNO die erste Spalte ist, DNAME die zweite Spalte, und LOC die dritte. Es gibt auch eine alternative mathematische Formalisierung, in der die Tupel als Abbildungen von Spaltennamen auf Werte dargestellt werden. Dann ist die Reihenfolge der Spalten (von links nach rechts) aber nicht mehr repräsentiert. In der Realität (SQL) ist beides wichtig.

Theorie vs. Praxis (1)

Undefinierte Zeilen-Reihenfolge:

- Da Relationen Mengen von Tupeln sind, ist die Reihenfolge der Zeilen unbestimmt.

  Insofern ist die Illustration als Tabelle nicht ganz zutreffend: In einer Tabelle gibt es ja eine erste, zweite, u.s.w. Zeile.

- Man kann das Ergebnis einer Anfrage aber sortieren lassen (so erhält man in der Ausgabe eine bestimmte Reihenfolge der Zeilen).

  Wenn man in einer Anfrage keine Sortierung verlangt hat, darf das DBMS das Ergebnis in beliebiger Reihenfolge ausgeben. Diese Reihenfolge kann sich auch von einer DBMS-Version zur nächsten ändern.

Theorie vs. Praxis (2)

Undefinierte Zeilen-Reihenfolge, Forts.:

- Wenn man neue Tupel (Zeilen) zu einer Relation (Tabelle) hinzufügt, kann man nicht festlegen, an welcher Stelle sie gespeichert wird.

  Sie wird nicht immer ganz am Ende gespeichert, sondern "wo gerade Platz ist".

- Es vereinfacht also z.B. die Einfüge-Befehle, daß die Speicher-Reihenfolge undefiniert ist.

  Man müßte sonst ja genau sagen, wo denn die neue Zeile eingefügt werden soll: Ganz am Anfang, ganz am Ende, oder nach/vor einer bestimmten Zeile. Das ist in SQL nicht nötig. Das DBMS bekommt so auch noch mehr Freiheiten, die die Effizienz verbessern.

Theorie vs. Praxis (3)

Unterschied (Duplikate):

- Im theoretisch vorgeschlagenen Relationenmodell sind Relationen Mengen von Tupeln, können also nicht das gleiche Tupel mehrfach enthalten.
- In SQL kann eine Anfrage dagegen Duplikate liefern. Bei Bedarf muß man explizit eine Duplikat-Eliminierung verlangen.

  Auch hier wäre das theoretische Modell einfacher, da nicht im einzelnen festgelegt werden muß, welche Duplikate bei einer bestimmten Anfrage und einem bestimmten DB-Zustand genau herauskommen. Es gibt aber bestimmte Anwendungen, bei denen Duplikate nützlich sind, außerdem spricht das Effizienz-Argument für Duplikate.

Schlüssel (1)

- Die Spalte DEPTNO ist als "Schlüssel" der Tabelle DEPT deklariert

  Schlüssel sind ein Typ von Integritätsbedingungen (Teil des Schemas).

- Das bedeutet, daß ein Wert für DEPTNO immer eine einzige Zeile der Tabelle eindeutig identifiziert.

DEPT		
DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Schlüssel (2)

- Z.B. enthält die Tabelle eine Zeile mit DEPTNO = 10.
- Würde man versuchen, eine weitere Zeile mit dem gleichen Wert 10 für DEPTNO einzufügen, so erhält man eine Fehlermeldung.
- Im Beispiel sind zufällig auch die Werte in den anderen beiden Spalten alle verschieden.
- Wenn im Datenbank-Schema für diese Spalten kein Schlüssel deklariert ist, würde das DBMS nicht verhindern, daß eine weitere Zeile mit bereits existierenden Werten in diesen Spalten eingefügt wird.

Schlüssel (3)

- Eine Tabelle kann mehrere Schlüssel haben:
  - ◇ Zum Beispiel könnte man im DB-Schema festlegen, daß sowohl die Abteilungsnummer (DEPTNO) eindeutig ist, als auch ihr Name (DNAME).
- Ebenso kann man die Kombination mehrerer Spalten zusammen als Schlüssel deklarieren:
  - ◇ Dann wären in den einzelnen Spalten Duplikate möglich, es darf nur keine Zeilen geben, die in dieser Kombination komplett übereinstimmen.

Schlüssel werden in Kapitel 6 und 8 ausführlich behandelt.

Zweite Beispiel-Tabelle (1)

EMP (Daten über Angestellte) hat folgende Spalten:

- EMPNO: Eindeutige Angestellten-Nummer.
- ENAME: Angestellten-Name.
- JOB: Beruf (z.B. MANAGER, SALESMAN).
- MGR: Nummer des direkten Vorgesetzten.
- HIREDATE: Einstellungsdatum.
- SAL: Gehalt.
- COMM: Provision (nur für Verkäufer).
- DEPTNO: Abteilung, in der der Angestellte arbeitet.

Zweite Beispiel-Tabelle (2)

EMP							
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Fremdschlüssel (1)

- Die Abteilungsnummer (DEPTNO) in der Tabelle EMP identifiziert eindeutig eine Zeile in der Tabelle DEPT.
- Sie stellt also einen Verweis auf die andere Tabelle dar (ein "logischer Zeiger"). Man kann die Daten der beiden Tabellen darüber verknüpfen.
- Die Spalte DEPTNO in EMP ist als Fremdschlüssel deklariert, der auf DEPT verweist.
- Das DBMS stellt dann sicher, daß nur Werte in diese Spalte eingetragen werden können, die auch tatsächlich in DEPT (in der Spalte DEPTNO) auftreten.

Fremdschlüssel (2)

- Fremdschlüssel sind auch eine Art von Integritätsbedingung und werden in Kapitel 8 ausführlicher behandelt.

  Die Verknüpfung von Tabellen in SQL wird in Kapitel 4 behandelt. Man kann Tabellen nicht nur über Fremdschlüssel verknüpfen, aber die Benutzung von Fremdschlüsseln ist der häufigste Fall.

- Aufgabe: Gibt es in dieser Tabelle außer DEPTNO noch einen Fremdschlüssel?

  Tipp: Fremdschlüssel müssen nicht unbedingt auf eine andere Tabelle verweisen und müssen nicht unbedingt genauso heißen wie die referenzierte Schlüssel-Spalte.

Nullwerte

- Das relationale Modell erlaubt es, daß Tabelleneinträge leer bleiben.

  Im Schema wird festgelegt, daß bestimmte Spalten nicht leer sein dürfen (wieder eine Art von Integritätsbedingung).

- Man sagt dann, die Spalte enthält einen "Nullwert".

  Der Nullwert ist nicht die Zahl 0. Er wird in Vergleichen besonders behandelt, siehe Kapitel 4.

- Nullwerte in der Beispiel-Tabelle:
  - ◇ Nur Verkäufer haben eine Provision.
  - ◇ Der Chef der Firma hat keinen Vorgesetzten.

Schema-Notation (1)

- Es gibt viele Notationen für relationale Schemata.

  Alle Details enthalten die `CREATE TABLE`-Anweisungen in SQL, sie sind aber etwas unübersichtlich. Siehe auch Kapitel 8.

- Man kann z.B. die Tabellen skizzieren, jeweils die Spalten des Schlüssels unterstreichen, und Fremdschlüssel durch Pfeile kennzeichnen:

EMP			
$\underline{\texttt{EMPNO}}$	ENAME	...	DEPTNO

DEPT		
$\underline{\texttt{DEPTNO}}$	DNAME	LOC

Einige Autoren zeichnen die Pfeile in umgekehrter Richtung.

Schema-Notation (2)

- Kurznotation in dieser Vorlesung:
  - ◇ Man schreibt den Tabellen-Namen auf, und dahinter in Klammern die Spalten, z.B.

    DEPT($\underline{\texttt{DEPTNO}}$, DNAME, LOC)

  - ◇ Schlüssel: Durch Unterstreichen markiert.

    Man kann so nur einen Schlüssel markieren ("Primärschlüssel"): Falls mehrere Spalten unterstrichen: Kombination ist Schlüssel.

  - ◇ Fremdschlüssel: Pfeil und referenzierte Tabelle:

    EMP($\underline{\texttt{EMPNO}}$, ENAME, ..., COMM$^{\circ}$, DEPTNO→DEPT)

  - ◇ Spalten mit Nullwerten: Markiert mit $^{\circ}$ oder ?.

Inhalt

1. Relationales Modell, Beispiel-Datenbank
2. Benutzung von SQL*Plus
3. Einfache SQL-Anfragen
4. Historische Bemerkungen

Web-Schnittstelle (1)

- Die einfachste Art, SQL-Anfragen auszuprobieren, ist folgendes Web-Formular zu benutzen:

  `http://mozart.informatik.uni-halle.de:8088/oradb/sql_up.html`

- Man kann SQL-Anfragen eingeben, auf "Ausführen" klicken, und bekommt das Ergebnis angezeigt.
- Folgende Tabellen stehen zur Verfügung:
  - ◇ Angestellten-Datenbank: `EMP`, `DEPT`, ...
  - ◇ CD-Datenbank (siehe Übung):

    `KOMPONIST`, `STUECK`, `AUFNAHME`, `CD`, `SOLIST`.

Web-Schnittstelle (2)

- Warum reicht das noch nicht?
  - ◇ Keine eigenen Tabellen.
  - ◇ Keine Updates.
- Daher erhält jeder Teilnehmer einen eigenen "Account" (Benutzerkennung) in einer Oracle-DB.
- Man kann dann z.B. das Programm SQL*Plus benutzen, um SQL-Anweisungen einzugeben.

  Während SQL*Plus eine rein textorientierte Schnittstelle hat, gibt es inzwischen auch das "SQL*Plus Worksheet", das etwas graphische Schnittstelle "um SQL*Plus herum" bietet (siehe Folie 2-53).

Oracle und SQL*Plus (1)

- SQL ist die Standard-Datenbanksprache für relationale Datenbank-Managementsysteme (RDBMS).
- Oracle "versteht" / "unterstützt" SQL, wie alle anderen modernen RDBMS (z.B. DB2) auch.

  Die Systeme unterscheiden sich allerdings in vielen kleinen Details.
- SQL*Plus ist Oracle's interaktive Schnittstelle zur Datenbank (direkte Eingabe von SQL-Befehlen).

  SQL*Plus ist nicht Oracle! Das DBMS selbst (der DB Server) läuft als Menge von (Hintergrund-)Prozessen, typischerweise auf einer anderen Maschine im Netz. SQL*Plus läuft auf dem Client (lokaler Rechner).

Oracle und SQL*Plus (2)

- Wenn man ein Anwendungsprogramm z.B. in PHP, Java, oder C schreibt, so kommuniziert das mit dem Oracle Server auch in SQL.
- SQL*Plus ist daran aber nicht beteiligt.

  Man kann SQL*Plus also als ein spezielles Anwendungsprogramm verstehen, das von Oracle direkt mitgeliefert wird (Werkzeug). Man könnte ein ähnliches Programm auch selbst schreiben.
- Die Unterscheidung ist auch deswegen wichtig, weil es spezielle SQL*Plus Befehle gibt (zusätzlich zu SQL). Diese können in Anwendungsprogrammen dann nicht genutzt werden.

Oracle und SQL*Plus (3)

- Die Hauptaufgabe von SQL*Plus ist
  - ◇ SQL-Anweisungen vom Benutzer entgegen zu nehmen,

    Es hat einige Editier-Funktionen, um z.B. Fehler zu korrigieren.
  - ◇ die Anweisung an den Server zu schicken, und das Ergebnis vom Server zu empfangen, und
  - ◇ die Ausgabe (Ergebnis-Tabelle) zu drucken.

    Es bietet einige Möglichkeiten, die Ausgabe-Formatierung zu steuern (z.B. wie breit die einzelnen Spalten sein sollen, wo bei Bedarf Zeilenumbrüche gemacht werden). Einfachere "Reports" (gedruckte Zusammenstellungen/Übersichten von Daten) lassen sich ganz in SQL*Plus entwickeln.

Oracle und SQL*Plus (4)

- SQL*Plus kann auch Batch-Dateien (Skripte) ausführen, das sind Dateien, die eine Folge von SQL und SQL*Plus-Befehlen enthalten.

  Es gibt dazu einen Ersetzungsmechanismus für Variablen/Parameter.

- Wenn man die gleichen Anfragen mehrfach stellen möchte (ggf. mit unterschiedlichen Werten an bestimmten Stellen), kann man sie in eine Datei schreiben und von SQL*Plus ausführen lassen.

  Einfache Anwendungsprogramme muß man also nicht unbedingt in Java, PHP, C, u.s.w. schreiben, sondern kann sie auch "in SQL*Plus" schreiben.

SQL*Plus: Einloggen (1)

- Unter UNIX und Linux gibt man das Kommando "`sqlplus`" ein, um SQL*Plus zu starten.

  Einige Umgebungsvariablen müssen gesetzt sein: `ORACLE_HOME` (z.B. auf `/oracle/OraHome1/`), `ORACLE_SID` (DB-Name, z.B. `studdb`), u.U. auch `ORACLE_BASE` (z.B. `/oracle/OraHome1/`), `TWO_TASK` (z.B. `studdb`). Außerdem muß man Oracle in `PATH` und ggf. `LD_LIBRARY_PATH` aufnehmen. Mit `NLS_LANG` kann man eine Sprache (z.B. `GERMAN_GERMANY.WE8ISO8859P1`).

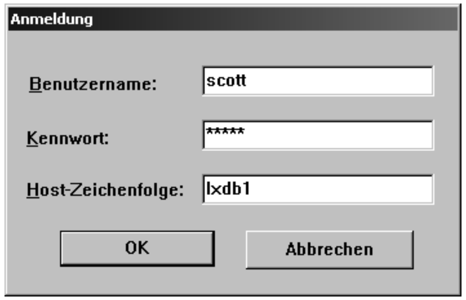
- Unter Windows: SQL*Plus im "Start" Menü.

  Z.B. `Start` → `Alle Programme` → `Datenbanken` → `Oracle` → `OraClient10g` → `Anwendungsentwicklung` → `SQL*Plus`

- Man muß sich dann bei der Datenbank anmelden ("einloggen").

SQL*Plus: Einloggen (2)

- SQL*Plus fragt dann nach Benutzername (Login, Account) und Passwort (geheimes Kennwort).



Mit der "Host-Zeichenfolge" wählt man die DB (z.B. `studdb`). Das ist auf Folie 2-37 näher erläutert.

SQL*Plus: Einloggen (3)

- Oracle hat eine eigene Benutzerverwaltung.
- Benutzername und Passwort für das Betriebssystem (Windows, Linux, UNIX) funktionieren also nicht automatisch auch für Oracle.
- Die verschiedenen DBMS unterscheiden sich hier: Manche übernehmen die Benutzer auch direkt vom Betriebssystem.
- Wenn man in Oracle/SQL*Plus angemeldet ("eingelogged") ist, kann man das Passwort mit dem Kommando "`password`" ändern.

SQL*Plus: Einloggen (4)

- Im Netz kann es mehrere Oracle-Datenbanken geben. Mit welcher man sich verbinden will, kann man mit der "Host-Zeichenfolge" festlegen.

  Bei einer lokalen Datenbank kann man dieses Feld normalerweise frei lassen. In einer Konfigurationsdatei sind die Host-Strings auf die genauen Netzwerk-Adressen und Datenbank-Namen abgebildet: `C:\Programme\Oracle\client_10.2.0\NETWORK\ADMIN\tnsnames.ora`

- Wenn man SQL*Plus aus der Kommandozeile startet, kann man die "Host-Zeichenfolge" an den Benutzernamen anhängen (durch "`@`" getrennt):

```
UNIX> sqlplus scott@studdb
```


Benutzer und DB-Schemata

- In Oracle hat jeder Benutzer ein eigenes Schema.

  Jeder Benutzer (Account) hat ein und nur ein Datenbank-Schema, und jedes Schema gehört genau einem Benutzer. Es besteht also eine 1:1 Beziehung zwischen Benutzern und Schemata, so daß meist von Benutzern geredet wird und nur selten explizit von einem Schema.

- Verschiedene Benutzer können Tabellen mit gleichem Namen haben.

  Diese Tabellen sind verschieden, weil sie zu unterschiedlichen Schemata (Namensräumen) gehören. In der Oracle Datenbank werden Tabellen global identifiziert über den Benutzernamen ihres "Besitzers" (Schema) und den Tabellen-Namen. Z.B. kann man auf die Tabelle `presidents` des Nutzers `brass` als "`brass.presidents`" zugreifen (wenn ihr Besitzer `brass` entsprechende Zugriffsrechte vergeben hat).

Beispiel-Tabellen

- Man kann eine Kopie der Beispieltabellen (`EMP` etc.) unter dem eigenen Benutzer-Account installieren:
  - ◇ Man rufe "`demobld`" vom UNIX/Windows Kommandoprompt auf.

    Dieses Programm probiert erst einen Default-Account (basierend auf dem Betriebssystem-Login). Wenn das nicht klappt, druckt es eine Fehlermeldung und fragt nach dem tatsächlichen Account.

  - ◇ Oder man gibt Folgendes in SQL*Plus ein:

    "`@D:\Software\Oracle9i\sqlplus\demo\demobld`"

    Der genaue Pfad `D:\...` hängt von der Installation ab. Die Datei `demobld.sql` enthält SQL-Anweisungen, die die Tabellen anlegen und mit Daten füllen. Allgemein führt das Kommando "`@`x" die Anweisungen in x`.sql` aus.

Benutzung von SQL*Plus (1)

- Der SQL*Plus Prompt ist "`SQL>`".

  "Prompt" heißt auch "Eingabeaufforderung". Man erkennt daran, daß das Programm (und welches) bereit ist, eine Eingabe entgegen zu nehmen. Zum Teil ändert sich der Prompt je nach Zustand des Programms. So druckt SQL*Plus bei Fortsetzungszeilen "2", "3", etc.

- In SQL*Plus müssen SQL-Anfragen (auch Updates etc.) mit "`;`" beendet werden.

  Alternative: "/" allein auf einer Zeile.

- Dadurch können sich SQL-Anfragen über mehrere Zeilen erstrecken.

  Man muß SQL*Plus kenntlich machen, wann die Anfrage vollständig ist und ausgeführt werden soll.

Benutzung von SQL*Plus (2)

- Das ";" ist formal nicht Teil der SQL-Anfrage.

  Es ist nur die Ende-Markierung, die SQL*Plus nach der eigentlichen Anfrage benötigt. In den graphischen Schnittstellen von DB2 und SQL Server muß der Benutzer stattdessen auf "ausführen" klicken, wenn die Anfrage vollständig ist. Dort wäre ";" zum Teil sogar ein Syntaxfehler.

- Um SQL*Plus zu verlassen, gebe man "`exit`" oder "`quit`" ein.

  Im Gegensatz zu den SQL-Anweisungen brauchen diese zusätzlichen SQL*Plus Kommandos kein Semikolon am Ende. Wenn sie sich einmal über mehrere Zeilen erstrecken sollen, muß man jede Zeile (außer der letzten) mit einem Bindestrich "-" beenden.

Benutzung von SQL*Plus (3)

- Man muß klar unterscheiden zwischen
  - ◇ SQL Anweisungen (Anfragen, Updates, etc.),
  - ◇ SQL*Plus Kommandos (z.B. `exit`, `l`, `r`, `@`x).
- SQL Anweisungen können sich über mehrere Zeilen erstrecken und werden in SQL*Plus mit ";" beendet. SQL*Plus Kommandos sind dagegen einzeilig (notfalls "-").
- Wenn man auf Oracle aus eigenen Anwendungsprogrammen zugreift, sind die SQL*Plus Erweiterungen nicht verfügbar (auch nicht nötig).

Benutzung von SQL*Plus (4)

- Der Inhalt einer Tabelle, z.B. `DEPT`, kann auf folgende Art ausgegeben werden (erste SQL-Anfrage!):

  ```
  SELECT * FROM DEPT;
  ```

- Die Tabelle (Sicht) `CAT` ("catalog") des Systemkatalogs enthält alle Tabellen des aktuellen Nutzers:

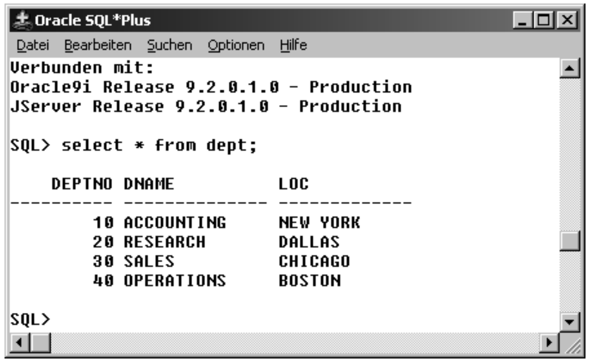
  ```
  SELECT * FROM CAT;
  ```

- Die Spalten einer Tabelle kann man auf folgende Art herausfinden:

  ```
  DESCRIBE EMP
  ```

  Dies ist ein SQL*Plus Kommando, daher braucht man kein ";".

Benutzung von SQL*Plus (5)



Benutzung von SQL*Plus (6)

```
UNIX> sqlplus
... (Version und Copyright Information für SQL*Plus)
Enter user-name: scott (oder scott@studdb)
Enter password: tiger (nicht sichtbar)
... (Version des Datenbank-Servers)
SQL> select * from dept;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```


SQL*Plus: Fehler, Editor (1)

- SQL*Plus hat einen "Puffer" (Zwischenspeicher) für die letzte SQL-Anweisung. Es kann geändert und neu ausgeführt werden.

  Z.B. im Fall eines Syntaxfehlers (wenn Oracle das Kommando nicht verstehen konnte, weil man sich nicht ganz exakt an die Grammatik der formalen Sprache SQL gehalten hat).

- `c/alt/neu` (`change`) ersetzt im Puffer das erste Vorkommen von "`alt`" durch "`neu`".

  In der aktuellen Zeile des Puffers (nach einem Fehler ist das die Zeile, in der Oracle den Fehler bemerkt hat — der eigentliche Fehler könnte aber auch an einer anderen Stelle sein).
  Man kann auch `c/alt/neu/` oder z.B. `c!old!new` schreiben.

SQL*Plus: Fehler, Editor (2)

- `l` (`list`) zeigt den Inhalt des Puffers.

  Die aktuelle Zeile ist dabei mit einem Sternchen "*" markiert.
  `l3` zeigt nur Zeile 3 und macht sie zur aktuellen Zeile.
  `i ...`: Nach aktueller Zeile neue Zeile ... einfügen.
  `3 ...`: Zeile 3 ersetzen durch ...

- `r` (`run`) führt den Inhalt des Puffers aus.
- Nur SQL-Anweisungen werden im Puffer gespeichert, nicht SQL*Plus Kommandos.

  Mit den SQL*Plus Kommandos wie `c` soll ja gerade der Puffer-Inhalt bearbeitet werden. Da SQL-Anweisungen am ersten Wort erkannt werden (z.B. `SELECT`, `INSERT`), muß man die Anweisung neu eingeben, wenn man sich schon im ersten Wort vertippt hat (sie wird nicht gepuffert, obwohl sie auch kein gültiges SQL*Plus-Kommando ist).

SQL*Plus: Fehler, Editor (3)

```
SQL> select *
  2  frm dept;
frm dept

ERROR at line 2:
ORA-00923: FROM keyword not found where expected
SQL> c/frm/from/
  2* from dept
SQL> r
  1  select *
  2* from dept
```

SQL*Plus: Fehler, Editor (4)

- Oracle meldet nur einen Fehler pro Anweisung. Es kann eventuell weitere Fehler geben.

  Was der Benutzer tatsächlich gemeint hat, und wo damit der oder die eigentlichen Fehler liegen, kann Oracle natürlich nicht wissen. Die Position des gemeldeten Fehlers ist leider auch nicht die erste Position, an der ein Fehler bemerkt werden könnte. Das ist heute in der Informatik eigentlich nicht mehr Stand der Technik.

- Hier fehlen z.B. die Anführungszeichen `'...'`:

```
SQL> insert into DEPT values(60, WWW, Dallas);
insert into DEPT values(40, WWW, Dallas)
                                  *
ERROR at line 1:
ORA-00984: column not allowed here
```


SQL*Plus: Fehler, Editor (5)

- `edit` schreibt den Inhalt des Puffers in eine Datei und ruft einen Editor dafür auf.

  Den Editor kann man wählen z.B. mit `define _editor = vi`. Diese Auswahl gilt aber nur für die aktuelle Sitzung, wird also beim Verlassen von SQL*Plus vergessen. Man kann sie aber in die Datei `login.sql` schreiben, die bei jedem Start von SQL*Plus gelesen wird (muß unter UNIX im aktuellen Verzeichnis beim Aufruf von SQL*Plus stehen).

- Man kann SQL-Anfragen auch in eine Datei schreiben, z.B. “`x.sql`”, und diese Datei dann mit `@x` ausführen (in SQL*Plus).

  Z.B. kann man einen Editor in einem Fenster offen haben, und dort die SQL-Anfrage entwickeln, abspeichern, und dann die SQL-Anfrage in einem anderen Fenster ausführen, in dem SQL*Plus läuft.

Mehr SQL*Plus Kommandos

- Beim Ausgeben von Tabellen druckt SQL*Plus für jede “Seite” Spalten-Überschriften. Normalerweise sind die Seiten recht klein. Z.B.: “`set pagesize 50`”.

  Man probiere auch “`help set`”, “`show pagesize`” und “`show all`”.

- `spool x` schreibt alle Ein- und Ausgaben (bis zu `spool off`) in die Datei `x.lst`.

  “`spool off`” ist nötig, vorher werden meist keine Daten geschrieben.

- Mit `column MGR format 9999` kann man die Ausgabebreite für die Spalte `MGR` auf 4 setzen.

  Für Zeichenketten z.B. `column X format A20 word_wrapped`

SQL*Plus: Parameter

- In SQL*Plus wird das Zeichen “&” benutzt, um Parameter zu markieren, die durch Eingaben des Benutzers zu ersetzen sind:

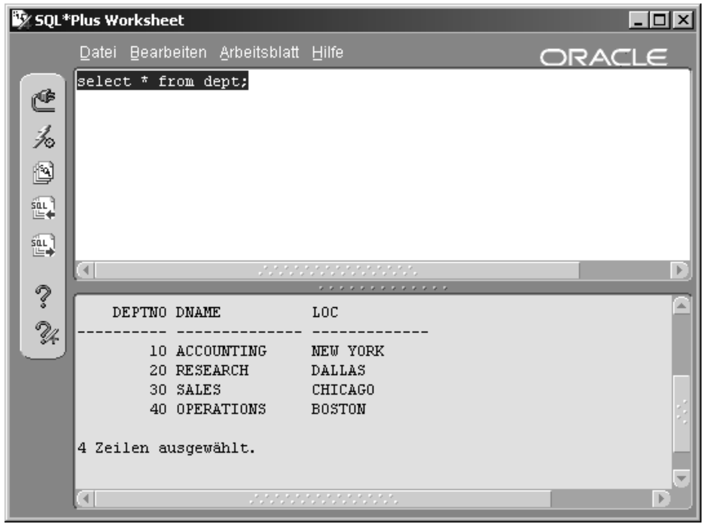
```
SQL> insert into DEPT
  2  values (50, 'R&D', 'Pittsburgh');
Enter value for d:
```

- Falls man “&” eingeben will, muß man erst ein “Escape”-Zeichen vereinbaren, z.B. “`set escape \`”, und dieses Zeichen dem “&” voranstellen: “`R\&D`”.

  Werte für Parameter können auch mit `define` definiert werden. Eine hübschere Benutzerabfrage gibt es mit `accept`, siehe `help accept`.

SQL*Plus Worksheet

- Inzwischen gibt es das SQL*Plus Worksheet als graphische Schnittstelle.
- Das Fenster (siehe nächste Folie) besteht aus zwei Teilen:
  - ◇ In der oberen Hälfte gibt man die Anweisung ein.
  - ◇ Unten wird das Ergebnis angezeigt.
- Durch Klicken auf den "Blitz"-Knopf wird die Anweisung ausgeführt.



Inhalt

1. Relationales Modell, Beispiel-Datenbank
2. Benutzung von SQL*Plus
3. Einfache SQL-Anfragen
4. Historische Bemerkungen

Einfache SQL-Anfragen (1)

- Einfache SQL-Anfragen haben die Struktur:

```
SELECT ... FROM ... WHERE ...
```

- Nach `FROM` gibt man die Tabelle(n) an, auf die sich die Anfrage bezieht.
- Nach `WHERE` gibt man Bedingungen für die gesuchten Zeilen an.

  Wenn man die `WHERE`-Klausel wegläßt, werden alle Zeilen ausgewählt.
- Nach `SELECT` gibt man an, welche Spalten gedruckt werden sollen ("*" heißt: alle Spalten).

Einfache SQL-Anfragen (2)

- Z.B.: Drucke die gesamte Tabelle "DEPT":

```
SELECT * FROM DEPT
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- Folgende Anfrage hat genau das gleiche Ergebnis:

```
SELECT DEPTNO, DNAME, LOC FROM DEPT
```


Einfache SQL-Anfragen (3)

- Die Groß- und Kleinschreibung spielt bei SQL keine Rolle, außer innerhalb von Zeichenketten ('...').

  Eine weitere Ausnahme sind "Delimited Identifier", die in "..." eingeschlossen sind (braucht man nur, wenn Spaltennamen unbedingt Kleinbuchstaben oder Sonderzeichen enthalten sollen).

- SQL ist eine format-freie Sprache.

  Man kann Zeilenumbrüche, Leerzeichen u.s.w. beliebig zwischen den Wortsymbolen ("token") einfügen.

- Man kann die letzte Anfrage auch so schreiben:

```
select deptno, DName ,   loc
from   dept
```


Einfache Bedingungen (1)

- Alle Daten der Abteilung in "DALLAS":

```
SELECT * FROM DEPT WHERE LOC = 'DALLAS'
```

DEPTNO	DNAME	LOC
20	RESEARCH	DALLAS

- Zeichenketten (Strings) werden mit '...' markiert.
- In Zeichenketten ist die Groß- und Kleinschreibung wichtig (je nach DBMS und Konfiguration). Folgende Anfrage liefert also 0 Zeilen (leere Menge):

```
SELECT * FROM DEPT WHERE LOC = 'Dallas'
```


Einfache Bedingungen (2)

- Drucke Namen, Beruf und Gehalt ("salary") aller Angestellten, die mindestens $2500 verdienen:

```
SELECT ENAME, JOB, SAL
FROM   EMP
WHERE  SAL >= 2500
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

- Zahlen werden nicht in '...' eingeschlossen!

Einfache Bedingungen (3)

- Man muß Spalten, die in Bedingungen verwendet werden, nicht unbedingt ausgedrucken.

  WHERE wird vor SELECT ausgewertet.

- Z.B.: Drucke Angestelltennummer und Name aller Manager:

```
SELECT EMPNO, ENAME
FROM   EMP
WHERE  JOB = 'MANAGER'
```

EMPNO	ENAME
7566	JONES
7698	BLAKE
7782	CLARK

Einfache Bedingungen (4)

- Vergleichsoperatoren in SQL:

  = : "ist gleich"

  <> : "ist verschieden von"

  < : "ist kleiner als"

  <= : "ist kleiner oder gleich"

  > : "ist größer als"

  >= : "ist größer oder gleich"

- Diese Vergleichsoperatoren können sowohl für Zahlen als auch für Zeichenketten verwendet werden.

  Vergleiche zwischen Zahlen und Zeichenketten sind aber nicht gestattet oder können zu überraschenden Ergebnissen führen.

Muster-Vergleich (1)

- SQL hat auch einen Operator für den Mustervergleich von Zeichenketten (erlaubt "wildcards").
- Z.B.: Drucke Nummer und Namen aller Angestellen, deren Name mit "S" beginnt.

```
SELECT ENAME
FROM   EMP
WHERE  ENAME LIKE 'S%'
```

EMPNO	ENAME
7369	SMITH
7788	SCOTT

Muster-Vergleich (2)

- "%" ist ein "wildcard" (Joker-Zeichen) und passt auf eine beliebige Zeichenkette.
- "_" passt entsprechend auf ein einzelnes beliebiges Zeichen.

  Sonst in der Informatik üblich: "*" statt "%", "?" statt "_".

- Man muß LIKE für den Mustervergleich verwenden, bei = haben % und _ keine spezielle Bedeutung.
- Wenn "S" irgendwo im Namen stehen soll (nicht unbedingt am Anfang):

```
ENAME LIKE '%S%'
```

Logische Verknüpfungen (1)

- Man kann `AND` ("und"), `OR` ("oder"), `NOT` ("nicht") und Klammern "(", ")" verwenden, um kompliziertere Bedingungen zu formulieren.
- Z.B.: Drucke Name und Gehalt aller Manager und des Präsidenten der Firma:

```
SELECT ENAME, SAL
FROM   EMP
WHERE  JOB = 'MANAGER' OR JOB = 'PRESIDENT'
```

- Diese Anfrage würde nicht mit `AND` funktionieren.

  Das Ergebnis wäre leer ("0 rows selected"), da `JOB` nicht gleichzeitig "`MANAGER`" und "`PRESIDENT`" sein kann.

Logische Verknüpfungen (2)

- Die `WHERE`-Bedingung wird für jede Zeile der unter `FROM` genannten Tabelle ausgewertet. Ist das Ergebnis "wahr" werden die `SELECT`-Spalten gedruckt.
- `AND` ist wahr, wenn beide Teile wahr sind, `OR` ist schon wahr, wenn nur ein Teil wahr ist:

C1	C2	C1 AND C2	C1 OR C2	NOT C1
falsch	falsch	falsch	falsch	wahr
falsch	wahr	falsch	wahr	wahr
wahr	falsch	falsch	wahr	falsch
wahr	wahr	wahr	wahr	falsch

Logische Verknüpfungen (3)

EMP				
EMPNO	ENAME	JOB	JOB='MANAGER'	JOB='PRESIDENT'
7369	SMITH	CLERK	falsch	falsch
7499	ALLEN	SALESMAN	falsch	falsch
7521	WARD	SALESMAN	falsch	falsch
7566	JONES	MANAGER	wahr	falsch
7654	MARTIN	SALESMAN	falsch	falsch
7698	BLAKE	MANAGER	wahr	falsch
7782	CLARK	MANAGER	wahr	falsch
7788	SCOTT	ANALYST	falsch	falsch
7839	KING	PRESIDENT	falsch	wahr
7844	TURNER	SALESMAN	falsch	falsch
⋮	⋮	⋮	⋮	⋮

Logische Verknüpfungen (4)

- Ohne Klammern bindet `AND` stärker als `OR` (und `NOT` hat die höchste Priorität).

  Bindungsstärken werden in Kapitel 4 genauer erklärt. Im Moment setze man zur Sicherheit Klammern, dann kann nichts schief gehen.

- Beispiel:

```
SELECT ENAME, SAL      Falsch!
FROM   EMP
WHERE  JOB = 'MANAGER' OR JOB = 'PRESIDENT'
AND    SAL >= 3000
```

- Das DBMS versteht die Bedingung als:

```
WHERE JOB = 'MANAGER'
OR    (JOB = 'PRESIDENT' AND SAL >= 3000)
```

Rechnen in SQL

- Man kann Spalten und/oder Konstanten mit den vier Grundrechenarten verknüpfen.
- Z.B. Jahresgehalt aller Manager:

```
SELECT ENAME, SAL * 12
FROM   EMP
WHERE  JOB = 'MANAGER'
```

ENAME	SAL * 12
JONES	35700
BLAKE	34200
CLARK	29400

- Zeichenketten können in den meisten Systemen mit dem Operator || verknüpft (konkateniert) werden.

  Viele Systeme bieten eine große Anzahl von Funktionen für Zahlen und Zeichenketten, siehe Kapitel 8.

Duplikat-Eliminierung (1)

- Anfragen können u.U. die gleiche Ergebnis-Zeile mehrfach liefern.
- Z.B.: "Gib die Berufsbezeichnungen der Angestellten von Abteilung 30 aus."

```
SELECT JOB
FROM   EMP
WHERE  DEPTNO = 30
```

JOB
SALESMAN
SALESMAN
SALESMAN
MANAGER
SALESMAN
CLERK

Duplikat-Eliminierung (2)

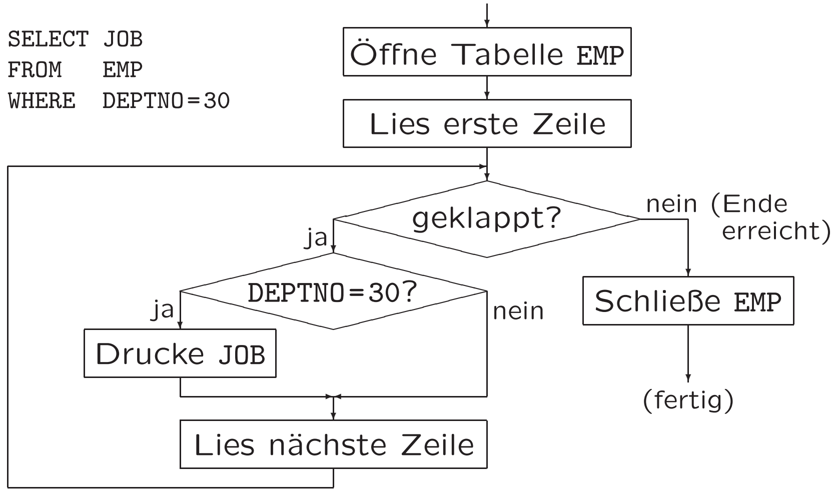
- Oft sind Duplikate in der Ausgabe nicht erwünscht (das Ergebnis ist unübersichtlich, besonders wenn die Ausgabe größer wird).

  Ggf. Beruf mit Häufigkeit: Mit COUNT/GROUP BY (siehe Kap. 5).
- Warum gibt es überhaupt Duplikate? Die Anfrage wird typischerweise ausgeführt, indem die Tabelle Zeile für Zeile durchlaufen wird (in einer "Schleife"), und für jede Zeile, die DEPTNO = 30 erfüllt, die Spalte "JOB" ausgegeben wird.

  Beispiel für interne Auswertung: Flußdiagramm auf nächster Folie. Ggf. schnellere Auswertung mit speziellen Datenstrukturen (Index).

Duplikat-Eliminierung (3)

```
SELECT JOB
FROM   EMP
WHERE  DEPTNO=30
```



Duplikat-Eliminierung (4)

- Schreibt man “`SELECT DISTINCT`” statt “`SELECT`”, so werden Duplikate eliminiert:

```
SELECT DISTINCT JOB
FROM   EMP
WHERE  DEPTNO = 30
```

JOB
CLERK
MANAGER
SALESMAN

- Auch wenn man mehrere Spalten ausgibt, schreibt man `DISTINCT` nur einmal:

```
SELECT DISTINCT DEPTNO, JOB
FROM   EMP
```

Der Grund ist, daß sich `DISTINCT` auf ganze Zeilen bezieht, die also in allen Spalten identisch sind. Nur partiell gleiche Zeilen sind wichtig.

Sortierung (1)

- Die Reihenfolge, in der die Ausgabezeilen gedruckt werden, ist nicht vorhersehbar, wenn man nicht explizit eine Sortierung verlangt (mit `ORDER BY`).

Wenn man nicht `ORDER BY` verwendet, schreibt der SQL-Standard nur vor, welche Tupel (mit ggf. welcher Anzahl von Duplikaten) gedruckt werden müssen. Die Reihenfolge darf der DBMS-Hersteller wählen.

- Z.B. Angestellte von Abteilung 20 mit Gehalt, sortiert nach Namen:

```
SELECT   ENAME, SAL
FROM     EMP
WHERE    DEPTNO = 20
ORDER BY ENAME
```

ENAME	SAL
ADAMS	1100
FORD	3000
JONES	2975
SCOTT	3000
SMITH	800

Sortierung (2)

- Beispiel: Gleiche Anfrage, aber Sortierung nach Gehalt, und zwar in umgekehrter Reihenfolge (von groß nach klein: “descending” / “absteigend”):

```
SELECT   ENAME, SAL
FROM     EMP
WHERE    DEPTNO = 20
ORDER BY SAL DESC
```

ENAME	SAL
SCOTT	3000
FORD	3000
JONES	2975
ADAMS	1100
SMITH	800

- Ohne das Schlüsselwort `DESC` bekommt man aufsteigende Sortierung (von kleinen zu großen Werten).

Wenn man will, kann man `ASC` schreiben, aber das ändert nichts.

Sortierung (3)

- Die Reihenfolge der Zeilen für `SCOTT` und `FORD` ist so nicht festgelegt, weil sie sich in dem Sortierkriterium (`SAL`) nicht unterscheiden (beide $ 3000).
- Für solche Fälle kann man auch ein zweites Sortierkriterium (u.s.w.) festlegen, das nur relevant ist, wenn die Sortierkriterien davor keine Entscheidung bringen:

```
SELECT   ENAME, SAL
FROM     EMP
WHERE    DEPTNO = 20
ORDER BY SAL DESC, ENAME
```

ENAME	SAL
FORD	3000
SCOTT	3000
JONES	2975
ADAMS	1100
SMITH	800

Aufgaben (1)

EMP							
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Aufgaben (2)

Formulieren Sie die folgenden Anfragen in SQL:

- Wer hat den Angestellten mit der Nummer 7839 (King) als direkten Vorgesetzen?
- Wer verdient zwischen $1000 und $2000? Drucken Sie Nummer, Name und Gehalt der Angestellten und sortieren Sie das Ergebnis nach der Angestelltennummer.

  "Zwischen" soll hier inklusive der Grenzen bedeuten.
- Welche Angestellten-Namen bestehen aus genau vier Buchstaben?

Aufgaben (3)

- Drucken Sie Name, Gehalt und Abteilungsnummer aller Angestellten, die in Abteilung 10 oder 30 arbeiten und weniger als $1500 verdienen.

  Stellen Sie sicher, daß wirklich beide Bedingungen erfüllt sind (Abteilung und Gehalt).
- Drucken Sie jeweils die Nummer des Vorgesetzten zusammen mit dem Beruf des Angestellten aus (d.h. welcher Chef hat welche Berufe in seinem Team). Drucken Sie jede Kombination nur einmal, und sortieren Sie nach der Nummer des Vorgesetzten, und bei gleicher Nummer nach dem Beruf.

Inhalt

1. Relationales Modell, Beispiel-Datenbank
2. Benutzung von SQL*Plus
3. Einfache SQL-Anfragen
4. Historische Bemerkungen

Relationales Modell (1)

- Das relationale Modell wurde 1970 von Edgar F. Codd ("Ted Codd", 1923–2003) vorgeschlagen.

  Codd arbeitete zu der Zeit im IBM Forschungslabor in San Jose. Codd wurde 1981 mit dem Turing Price ausgezeichnet (einer Art Nobelpreis für Informatik). [http://www.research.ibm.com/resources/news/20030423_edgarpassaway.shtml]

- Es war das erste Modell, das theoretisch definiert war, bevor es implementiert wurde.

  Frühere Datenmodelle waren Abstraktionen von schon vorher existierenden Systemen. Das relationale Modell wurde zuerst teils als "unimplementierbar" kritisiert (relationale DBMS könnten niemals hinreichend schnell sein).

Relationales Modell (2)

- Erste Implementierungen (1976):
  - System R (IBM)

    In diesem sehr einflußreichen Forschungsprototyp wurde die Sprache SQL eingeführt (an deren Entwurf Codd nicht beteiligt war).
  - Ingres (Stonebraker, UC Berkeley).
- Erste kommerzielle Systeme: Oracle (1979), Ingres (1980?) und IBM SQL/DS (1981).
- Das relationale Modell ist noch "State of the Art" in der Industrie, allerdings sind alle großen DBMS-Hersteller schon "objekt-relational".

  Stonebraker: mit Postgres ein Vater objektrelationaler Technologie.

Relationales Modell (3)

Gründe für Erfolg:

- Viel einfacher als frühere Datenmodelle.
  Nur ein Konzept: Endliche Relation.

  Frühere Datenmodelle enthielten Verkettungen/Verzeigerungen von Datensätzen (z.B. "Netzwerkmodell").

- Leicht verständlich, auch für Nicht-Spezialisten.
  Relationen entsprechen Tabellen.
- Abstraktion bekannter "Dateien von Datensätzen".
- Mengenorientierte Operationen.

  In früheren Modellen mußte man in Schleifen von einem Datensatz zum nächsten navigieren.

Relationales Modell (4)

Gründe für den Erfolg, Fortsetzung:

- Deklarative Anfragesprache: Man muß nicht über effiziente Anfrageauswertung nachdenken.

  Man muß nur Bedingungen für die gewünschten Daten aufschreiben.

- Kompaktere Formulierungen.

  SQL-Anfragen sind wesentlich kürzer als entsprechende Abfrageprogramme in älteren Datenmodellen.

- Freiere Möglichkeiten zur Kombination von Daten.

  Auch über das hinaus, was beim ursprünglichen Schema-entwurf geplant war.

- Solide Grundlage: Mathematische Logik.

SQL: Bedeutung

- Heute ist SQL die einzige DB-Sprache für relationale DBMS (Industriestandard).

  Andere Sprachen wie QUEL sind ausgestorben (die Sprache QBE hat zumindest einige graphische Schnittstellen zu Datenbanken inspiriert (z.B. in Access) und die Sprache Datalog hat SQL:1999 beeinflusst und wird noch in der Forschung studiert und weiterentwickelt). Jedes kommerzielle RDBMS muß heute eine SQL-Schnittstelle haben. Es kann zusätzlich noch andere (z.B. graphische) Schnittstellen geben.

- SQL wird verwendet für:
  - ◇ Interaktive "Ad-hoc"-Befehle und
  - ◇ Anwendungsprogrammentwicklung (in andere Sprachen wie C, Java, HTML eingebettet).

SQL: Geschichte

- SEQUEL, eine frühere Version von SQL, wurde von Chamberlin, Boyce et al. bei IBM Research, San Jose (1974) entwickelt.

  SEQUEL steht für "Structured English Query Language" ("Strukturierte englische Anfragesprache"). Manche Leute Sprechen SQL noch heute auf diese Art aus. Der Name wurde aus rechtlichen Gründen geändert (SEQUEL war bereits eine eingetragene Marke). Codd war auch in San Jose, als er das relationale Modell erfand.

- SQL war die Sprache des System/R (1976/77).

  System/R war ein sehr einflussreicher Forschungs-Prototyp.

- Ersten SQL-unterstützenden kommerziellen Systeme waren Oracle (1979) und IBM SQL/DS (1981).

SQL: Standards (1)

- Erster Standard 1986/87 (ANSI/ISO).

  Das war sehr spät, weil es schon viele SQL-Systeme auf dem Markt gab. Der Standard war der "kleinste gemeinsame Nenner". Er enthielt im wesentlichen nur die Schnittmenge der SQL-Dialekte.

- Erweiterung für Fremdschüssel usw. '89 (SQL-89).

  Diese Version wird auch SQL-1 genannt. Alle kommerziellen Implementierungen unterstützen heute diesen Standard, aber jede hat viele Erweiterungen. Der Standard hatte 120 Seiten.

- Größere Erweiterung: SQL2 oder SQL-92 (1992).

  626 Seiten, aufwärts kompatibel zu SQL-1. Der Standard definiert drei Kompatibilitätsgrade: "Entry", "Intermediate", "Full" (+ später "Transitional" zwischen ersten beiden). Oracle 8.0 und SQL Server 7.0 sind "Entry Level"-kompatibel, höhere Konstrukte zum Teil.

SQL: Standards (2)

- Weitere große Erweiterung: SQL:1999.

  Zuerst wurde SQL:1999 als eine Vorgänger-Version des SQL3-Standards angekündigt, aber nun scheint es SQL3 zu sein. Bis 12/2000 erschienen die Bände 1–5 und 10 des SQL:1999-Standards. Sie haben zusammen 2355 Seiten. Anstelle von Levels definiert der Standard nun "Core SQL" und eine Vielzahl von Paketen von Sprach-Features.

- Wichtige neue Möglichkeiten in SQL:1999:
  - ◇ Nutzer-definierte Datentypen, strukturierte Typen.

    Man kann nun "distinct types" definieren, die Domains sehr ähnlich sind, aber ein Vergleich zwischen verschiedenen "distinct types" ist nun ein Fehler. Es gibt auch Typ-Konstruktoren "`ARRAY`" und "`ROW`" für strukturierte Attributwerte und "`REF`" für Zeiger auf Zeilen.

SQL: Standards (3)

- Wichtige neue Möglichkeiten in SQL:1999, Forts.:
  - ◇ OO-Fähigkeiten, z.B. Vererbung/Untertabellen.
  - ◇ Rekursive Anfragen.
  - ◇ Trigger, persistent gespeicherte Module.
- Kleinere Veränderung: SQL:2003.

  Größte Neuerungen: Unterstützung von XML, Verwaltung von externen Daten. Außerdem gibt es nun Generatoren für eindeutige Nummern und einen "`MULTISET`" type constructor. Ansonsten ist es nur die 2. Auflage des SQL:1999-Standards. Auch die OLAP-Elemente (On-Line Analytical Processing), die als Zusatz zum SQL:1999-Standard veröffentlicht wurden, sind nun im SQL:2003-Standard integriert.

SQL: Standards (4)

- Bände des SQL:2003-Standards:
  - ◇ Teil 1: Framework (SQL/Framework) [84 S.]
  - ◇ Teil 2: Foundation (SQL/Foundation) [1268 S.]
  - ◇ Teil 3: Call-Level Interface (SQL/CLI) [406 S.]

    Dies spezifiziert eine Menge von Prozeduren, die verwendet werden können, um SQL-Anweisungen auszuführen und die Ergebnisse einzulesen (ähnlich zu ODBC).
  - ◇ Teil 4: Persistent Stored Modules (SQL/PSM) [186 S.]

    Programmiersprache für gespeicherte Prozeduren. Es macht SQL berechnungsuniversell und ist ähnlich zu PL/SQL in Oracle oder Transact SQL in Microsoft SQL Server und Sybase.

SQL: Standards (5)

- Bände des SQL:2003-Standards, fortgesetzt:
  - ◇ Teil 9: Management of External Data (SQL/MED)
  - ◇ Teil 10: Object Language Bindings (SQL/OLB) [404 Seiten]

    Dies definiert, wie SQL-Befehle in Java-Programme eingebettet werden können.
  - ◇ Teil 11: Information and Definition Schemas (SQL/Schemas) [298 Seiten]

    Dies definiert ein Standard-Data-Dictionary (System-Katalog).

SQL: Standards (6)

- Bände des SQL:2003-Standards, fortgesetzt:
  - ◇ Teil 13: SQL Routines and Types using the Java Programming Language (SQL/JRT) [206 S.]

    Dies legt fest, wie Javas statistische Methoden in SQL-Statements verwendet werden können und wie Java-Klassen als strukturierte Typen in SQL verwendet werden können.
  - ◇ Teil 14: XML-Related Specifications (SQL/XML) [268 S.]
- Der offizielle Name von Teil n des Standards ist [INCITS/] ISO/IEC 9075-n-2003: Information technology — Database Languages — SQL.

SQL: Standards (7)

- Die fehlenden Teile des SQL:2003-Standards werden wahrscheinlich nie erscheinen.

  Teil 5 ist in SQL:1999 "object language bindings" beschrieben. Dies wurde in Teil 2 integriert. Teil 6 sollte die X/OPEN XA-Spezifikation (Transaktionen) behandeln. Dieser Teil wurde gelöscht. Teil 7 sollte SQL/Temporal spezifizieren. Es wurde gestoppt, weil das Komitee große Meinungsverschiedenheiten hatte. Teil 8 sollte die erweiterten Objekte/abstrakten Datentypen behandeln und wurde in Teil 2 integriert. Teil 12 soll Replication behandeln.

- Es gibt einen verwandten Standard ISO/IEC 13249: "SQL multimedia and application packages".

  Teil 1: Framework, Teil 2: Full-Text, Teil 3: Spatial, Teil 5: Still Image, Teil 6: Data Mining.